🏠 / GitPython Quick Start Tutorial

GitPython Quick Start Tutorial

Welcome to the GitPython Quickstart Guide! Designed for developers seeking a practical and interactive learning experience, this concise resource offers step-by-step code snippets to swiftly initialize/clone repositories, perform essential Git operations, and explore GitPython's capabilities. Get ready to dive in, experiment, and unleash the power of GitPython in your projects!

git.Repo

There are a few ways to create a `git.Repo` object

Initialize a new git Repo

```python
# $ git init <path/to/dir>

from git import Repo

repo = Repo.init(path_to_dir)
```

Existing local git Repo

```python
repo = Repo(path_to_dir)
```

Clone from URL

For the rest of this tutorial we will use a clone from https://github.com/gitpython-developers/QuickStartTutorialFiles.git

```python
# $ git clone <url> <local_dir>

repo_url = "https://github.com/gitpython-developers/QuickStartTutorialFiles.git"

repo = Repo.clone_from(repo_url, local_dir)
```

Trees & Blobs

Latest Commit Tree

```
tree = repo.head.commit.tree
```

Any Commit Tree

```
prev_commits = [c for c in repo.iter_commits(all=True, max_count=10)]  # last 10
commits from all branches
tree = prev_commits[0].tree
```

Display level 1 Contents

```
files_and_dirs = [entry for entry in tree]
files_and_dirs

# Output
# [<git.Commit "SHA1-HEX_HASH">,
# <git.Commit "SHA1-HEX_HASH">,
# <git.Commit "SHA1-HEX_HASH">]
```

```
files_and_dirs = [(entry, entry.name, entry.type) for entry in tree]
files_and_dirs

# Output
# [(< git.Tree "SHA1-HEX_HASH" >, 'Downloads', 'tree'),
#  (< git.Tree "SHA1-HEX_HASH" >, 'dir1', 'tree'),
#  (< git.Blob "SHA1-HEX_HASH" >, 'file4.txt', 'blob')]
```

Recurse through the Tree

```python
def print_files_from_git(root, level=0):
    for entry in root:
        print(f'{"-" * 4 * level}| {entry.path}, {entry.type}')
        if entry.type == "tree":
            print_files_from_git(entry, level + 1)
```

```python
print_files_from_git(tree)

# Output
# | Downloads, tree
# ----| Downloads / file3.txt, blob
# | dir1, tree
# ----| dir1 / file1.txt, blob
# ----| dir1 / file2.txt, blob
# | file4.txt, blob
```

Usage

Add file to staging area

```python
# We must make a change to a file so that we can add the update to git

update_file = 'dir1/file2.txt' # we'll use local_dir/dir1/file2.txt
with open(f"{local_dir}/{update_file}", 'a') as f:
    f.write('\nUpdate version 2')
```

Now lets add the updated file to git

```python
# $ git add <file>
add_file = [update_file]  # relative path from git root
repo.index.add(add_file)  # notice the add function requires a list of paths
```

Notice the add method requires a list as a parameter

Warning: If you experience any trouble with this, try to invoke `git` instead via repo.git.add(path)

Commit

```
# $ git commit -m <message>
repo.index.commit("Update to file2")
```

List of commits associated with a file

```
# $ git log <file>

# relative path from git root
repo.iter_commits(all=True, max_count=10, paths=update_file)  # gets the last 10
commits from all branches

# Outputs: <generator object Commit._iter_from_process_or_stream at 0x7fb66c186cf0>
```

Notice this returns a generator object

```
commits_for_file_generator = repo.iter_commits(all=True, max_count=10,
paths=update_file)
commits_for_file = [c for c in commits_for_file_generator]
commits_for_file

# Outputs: [<git.Commit "SHA1-HEX_HASH-2">,
# <git.Commit "SHA1-HEX-HASH-2">]
```

returns list of `Commit` objects

Printing text files

Lets print the latest version of *<local_dir> dir1/file2.txt*

```
print_file = 'dir1/file2.txt'
tree[print_file]  # the head commit tree

# Output <git.Blob "SHA1-HEX-HASH-1">
```

```
blob = tree[print_file]
print(blob.data_stream.read().decode())

# Output
# file 2 version 1
# Update version 2
```

Previous version of *<local_dir>/dir1/file2.txt*

```
commits_for_file = [c for c in repo.iter_commits(all=True, paths=print_file)]
tree = commits_for_file[-1].tree  # gets the first commit tree
blob = tree[print_file]

print(blob.data_stream.read().decode())

# Output
# file 2 version 1
```

Status

- Untracked files

    Lets create a new file

    ```python
    f = open(f'{local_dir}/untracked.txt', 'w')  # creates an empty file
    f.close()
    ```

    ```python
    repo.untracked_files
    # Output: ['untracked.txt']
    ```

- Modified files

    ```python
    # Let's modify one of our tracked files

    with open(f'{local_dir}/Downloads/file3.txt', 'w') as f:
        f.write('file3 version 2')  # overwrite file 3
    ```

    ```python
    repo.index.diff(None)  # compares staging area to working directory

    # Output: [<git.diff.Diff object at 0x7fb66c076e50>,
    # <git.diff.Diff object at 0x7fb66c076ca0>]
    ```

    returns a list of `Diff` objects

    ```python
    diffs = repo.index.diff(None)
    for d in diffs:
        print(d.a_path)

    # Output
    # Downloads/file3.txt
    ```

Diffs

Compare staging area to head commit

```python
diffs = repo.index.diff(repo.head.commit)
for d in diffs:
    print(d.a_path)


# Output
```

```python
# lets add untracked.txt
repo.index.add(['untracked.txt'])
diffs = repo.index.diff(repo.head.commit)
for d in diffs:
    print(d.a_path)

# Output
# untracked.txt
```

Compare commit to commit

```python
first_commit = [c for c in repo.iter_commits(all=True)][-1]
diffs = repo.head.commit.diff(first_commit)
for d in diffs:
    print(d.a_path)

# Output
# dir1/file2.txt
```